AI Attack: Learning to play a video game using visual inputs

Jerry Luo, Neha Narwal, Rajiv Krishnakumar

CS231n Spring 2017, Stanford University

Problem Statement

One of the overarching goals of the deep learning community is to create AI that can make optimal decisions based on sensory input. This can be done in a game scenario [1].

The goal is to make a computer agent play the "endless" mode of the Super Nintendo Game "Tetris Attack" using only visual inputs (i.e. in-game screenshots).

We train an agent with human gameplay data using a Deep Convolutional Neural Network and evaluate how it performs compared to variations of random play. We use similar and complementary methods to work that was done on Nintendo 64 games [2].

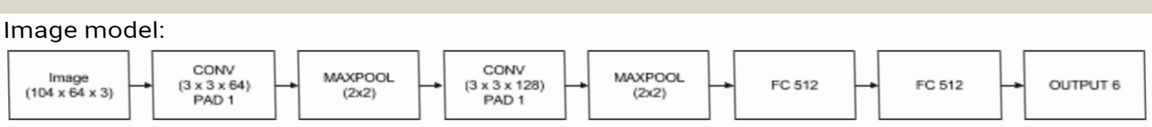
Methods

Once we've collected the frames and corresponding button press for that frame, we train a deep convolutional neural network to predict the next button press given an input frame. As an extension,we also trained a model that takes the current frame as well as the previous button press to predict the next button press in order to capture the memory of button presses.

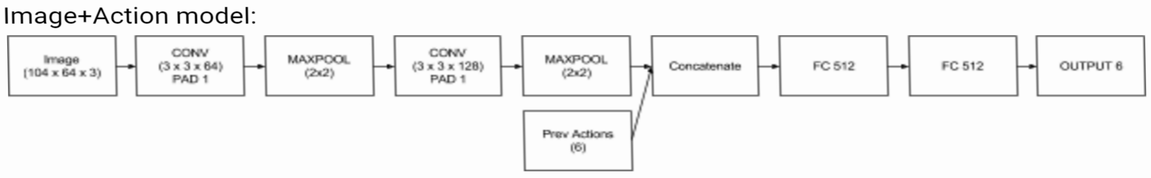
After training the predictive model, we let the agent play the game automatically using pyautogui and check the score that the agent received.

The agent's gameplay speed is bottlenecked by the screenshot capture utility, and the final gameplay speed is around 3 actions per second.

Image model:



Image+Action model:



* All CONV and FC layers are followed by a ReLU activation

Future Direction

Since overfitting indicates that we don't have enough data richness, the next step would be to gather more training data, perhaps with multiple people's gameplay to capture more variations.

We can also create saliency maps to try and better understand what features of the gameplay our models are capturing.

We can make the agent learn a Q function based on the current supervised learning policy, and use this to bootstrap Deep Q Learning [1].

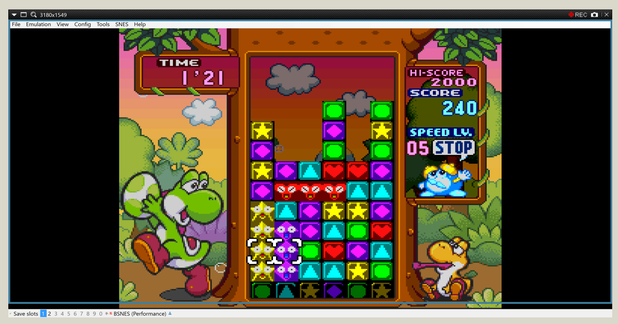
Datasets

We collected 60,000 frames with corresponding actions using the BizHawk emulator and Bandicam screen capture tool. We only capture a frame when an action occurs.

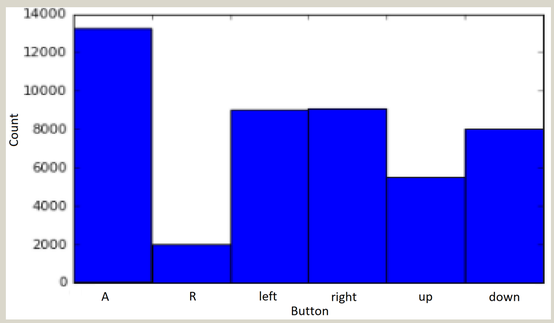
Frames are downsampled to 104 x 64 x 3 using bilinear interpolation.

The histogram below shows the uneven distrubution of button presses.

Occasionally the screen capture is delayed by a few frames causing our data to have some noise (i.e. inaccurate game state to button press correspondances).



Screen Capture process

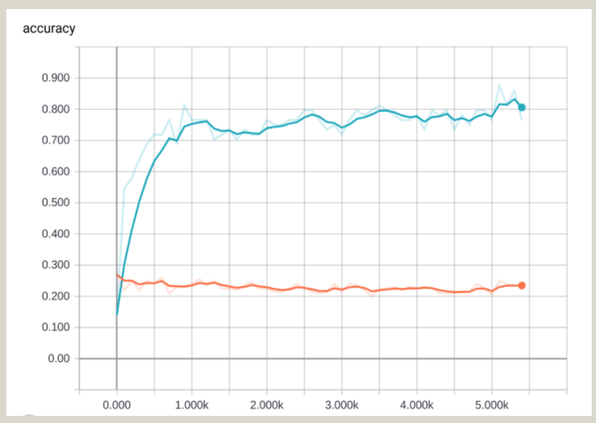


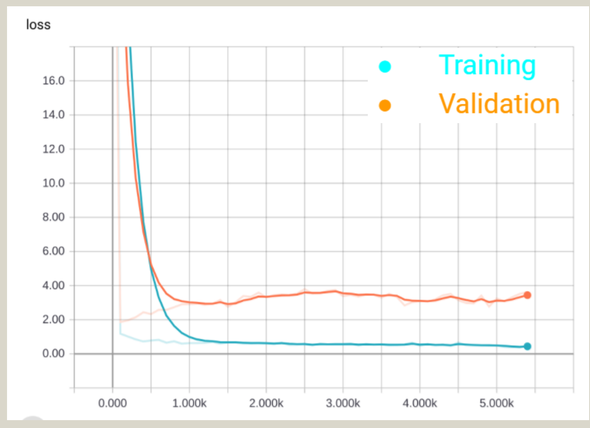
Histogram of button presses

Experimental Evaluation

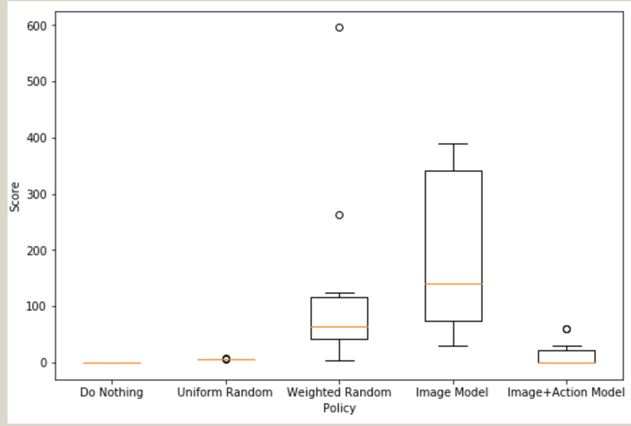
When training the models, we tend to overfit significantly, indicating that we lack data richness.

We use 0.5 dropout and 10^{-2} L2 regularization. At higher regularization, the validation accuracy doesn't improve, while the training accuracy gets worse.





We compare the scores over 10 runs for various control policies.



The image model does perform slightly better with more consistency than the weighted random policy despite low validation accuracy. Qualitatively, the agent tends to forget what it was last doing, and doesn't really follow any action trajectories.

The image model trained with the previous action doesn't do as well. Qualitatively, it does tend to follow an action trajectory better than the pure image model, but is oftentimes stuck in corners, which caused the poor performance.

[1] Mnih, Volodymyr, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. "Playing Atari with Deep Reinforcement Learning." [arXiv: 1312.5602]
[2] Chen, Zhao, and Darvin Yi. "The Game Imitation: Deep Supervised Convolutional Networks for Quick Video Game AI." [arXiv: 1702.05663]